

Textbox Documentation

The component provides a highly sophisticated text input element with glass morphism effects, advanced content projection system, multiple processing animations, and comprehensive Angular forms integration. It supports semantic variants, validation states, and extensive customization options.

How to use

Basic Usage

Simple textbox implementation with label, placeholder, and two-way data binding.

```
<aava-textbox
  label="Basic Input"
  placeholder="Enter text here"
  (change)="onTextboxChange($event)"
></aava-textbox>

---

onTextboxChange(event: Event) {
  const target = event.target as HTMLInputElement;
  console.log('Textbox value changed:', target.value);
}
```

Variants

The textbox component supports 6 semantic variants that control visual appearance and focus colors.

```

<aava-textbox
  label="Default Variant"
  placeholder="Default variant..."
  variant="default"
  (change)="onTextboxChange($event)"
></aava-textbox>

<aava-textbox
  label="Primary Variant"
  placeholder="Primary variant..."
  variant="primary"
  (change)="onTextboxChange($event)"
></aava-textbox>

<aava-textbox
  label="Success Variant"
  placeholder="Success variant..."
  variant="success"
  (change)="onTextboxChange($event)"
></aava-textbox>

<aava-textbox
  label="Error Variant"
  placeholder="Error variant..."
  variant="error"
  (change)="onTextboxChange($event)"
></aava-textbox>

<aava-textbox
  label="Warning Variant"
  placeholder="Warning variant..."
  variant="warning"
  (change)="onTextboxChange($event)"
></aava-textbox>

<aava-textbox
  label="Info Variant"
  placeholder="Info variant..."
  variant="info"
  (change)="onTextboxChange($event)"
></aava-textbox>

---

onTextboxChange(event: Event) {
  const target = event.target as HTMLInputElement;
  console.log('Textbox value changed:', target.value);
}

```

Available Variants

- default - Standard neutral appearance with brand primary focus
- primary - Primary variant with enhanced brand focus color
- success - Positive states and confirmations (green)
- error - Error states and validation failures (red)
- warning - Warning states and cautions (orange/yellow)

- info - Informational states and tips (blue)

Sizes

Five size options to accommodate different interface densities and layout requirements.

```
<aava-textbox
  label="Extra Small"
  placeholder="Extra Small..."
  size="xs"
></aava-textbox>

<aava-textbox label="Small" placeholder="Small..." size="sm"></aava-textbox>

<aava-textbox label="Medium" placeholder="Medium..." size="md"></aava-textbox>

<aava-textbox label="Large" placeholder="Large..." size="lg"></aava-textbox>

<aava-textbox
  label="Extra Large"
  placeholder="Extra Large..."
  size="xl"
></aava-textbox>
```

Available Sizes

- xs (Extra Small) - Extra small size for very compact interfaces
- sm (Small) - Small size for dense interfaces
- md (Medium) - Medium size for most use cases (default)
- lg (Large) - Large size for prominent inputs
- xl (Extra Large) - Extra large size for emphasis and accessibility

Icons & Affixes

Advanced content projection system supporting icons, prefixes, and suffixes through Angular's content projection.

```

<aava-textbox label="Amount" placeholder="0.00" type="number" (on)>
  <span slot="prefix">$</span>
  <span slot="suffix">USD</span>
</aava-textbox>

<aava-textbox
  label="Password"
  [type]="showPassword ? 'text' : 'password'"
  placeholder="Enter password"
>
  <aava-icon
    slot="icon-end"
    [iconName]="showPassword ? 'eye' : 'eye-off'"
    (click)="togglePasswordVisibility()"
  ></aava-icon>
</aava-textbox>

<aava-textbox
  label="With Icon Separator (Start)"
  placeholder="Search..."
  [iconSeparator]="true"
>
  <aava-icon slot="icon-start" iconName="search"></aava-icon>
</aava-textbox>

<aava-textbox
  label="With Icon Separator (End)"
  placeholder="Clear..."
  [iconSeparator]="true"
>
  <aava-icon slot="icon-end" iconName="x"></aava-icon>
</aava-textbox>

---

showPassword = false;

togglePasswordVisibility(): void {
  this.showPassword = !this.showPassword;
}

```

Content Projection Slots

- icon-start - Icons at the beginning of the input
- icon-end - Icons at the end of the input
- prefix - Text or elements before the input text
- suffix - Text or elements after the input text

Input Masking

Advanced input masking system powered by ngx-mask for formatted input patterns like phone numbers, dates, currency, and credit cards.

```

<!-- Phone Number Masking -->
<aava-textbox
  label="Phone Number"
  placeholder="(123) 456-7890"
  [mask]="maskPhone"
>
</aava-textbox>

<!-- Currency Masking -->
<aava-textbox
  label="Currency"
  placeholder="0.00"
  [mask]="maskCurrency"
  [maskThousandSeparator]="thousand"
  [maskDecimalMarker]="decimal"
>
  <span slot="prefix">$</span>
</aava-textbox>

<!-- Date Masking -->
<aava-textbox label="Date" placeholder="MM/DD/YYYY" [mask]="maskDate">
</aava-textbox>

<!-- Custom Pattern Masking -->
<aava-textbox
  label="Custom Pattern (AA-0000)"
  placeholder="AB-1234"
  [mask]="customMask"
  [maskPatterns]="customPatterns"
>
</aava-textbox>
<aava-textbox
  label="Phone Number (with Built-in Dropdown)"
  [placeholder]="currentCountryPlaceholder"
  [(ngModel)]="countryPhoneValue"
  [mask]="currentCountryMask" [
  prefixDropdown]="true"
  [prefixDropdownOptions]="countryOptions"
  [selectedPrefixOption]="selectedCountry"
  (prefixDropdownSelect)="onCountrySelect($event)"
>
</aava-textbox>

<aava-textbox
  label="Phone Number (with Custom Slot Dropdown)"
  [placeholder]="slotCurrentPlaceholder"
  [(ngModel)]="slotCountryPhoneValue"
  [mask]="slotCurrentMask"
  (clickOutSide)="closeSlotDropdown()"
>
  <!-- Custom dropdown projected into prefix slot -->
  <div slot="prefix" class="custom-country-dropdown">
    <div class="dropdown-trigger" (click)="toggleSlotDropdown()" tabindex="0" role="button"
      [attr.aria-expanded]="slotIsDropdownOpen">
      <span class="country-label">{{ slotSelectedCountry.label }}</span>
      <svg class="chevron-icon" [class.open]="slotIsDropdownOpen" width="16" height="16" viewBox="0 0 16 16"
        fill="none" stroke="currentColor">
        <polyline points="6 9 12 15 18 9"></polyline>
      </svg>
    </div>
  </div>

```

```

</div>
<div slot="dropdown" class="custom-country-dropdown">
  <div class="dropdown-menu" *ngIf="slotIsDropdownOpen" role="listbox" tabindex="-1">
    <div *ngFor="let option of countryOptions" class="dropdown-item"
      [class.selected]="option.value === slotSelectedCountryCode"
      (click)="selectSlotCountry(option); $event.stopPropagation()" (keydown.enter)="
        selectSlotCountry(option); $event.stopPropagation()"
      " (keydown.space)="
        selectSlotCountry(option); $event.stopPropagation()
      " tabindex="0" role="option" [attr.aria-selected]="option.value === slotSelectedCountryC
      {{ option.label }}>
    </div>
  </div>
</div>
</aava-textbox>

```

```

maskPhone = '(000) 000-0000';
maskCurrency = 'separator.2';
thousand: ',' | '' = ',';
decimal: '.' | ',' | ['.', ','] = '.';

maskDate = '00/00/0000';

customMask = 'SS-0000';
customPatterns: Record<string, { pattern: RegExp }> = {
  S: { pattern: /[A-Za-z]/ },
  '0': { pattern: /\d/ },
};

countryPhoneValue = '';
countryOptions = [
  {
    label: 'US +1',
    value: 'us',
    mask: '(000) 000-0000',
    placeholder: '(123) 456-7890',
  },
  {
    label: 'IN +91',
    value: 'in',
    mask: '00000 00000',
    placeholder: '98765 43210',
  },
  {
    label: 'UK +44',
    value: 'uk',
    mask: '00000 000000',
    placeholder: '20123 456789',
  },
];
selectedCountry = this.countryOptions[0];
get currentCountryMask(): string {
  return this.selectedCountry.mask;
}
get currentCountryPlaceholder(): string {

```

```

        return this.selectedCountry.placeholder;
    }
    onCountrySelect(option: { label: string; value: string }): void {
        const country = this.countryOptions.find((c) => c.value === option.value);
        if (country) {
            this.selectedCountry = country;
            this.countryPhoneValue = '';
        }
    }

    slotCountryPhoneValue = '';
    slotSelectedCountryCode = 'us';
    slotIsDropdownOpen = false;

    get slotSelectedCountry(): {
        return (
            this.countryOptions.find(
                (c) => c.value === this.slotSelectedCountryCode
            ) || this.countryOptions[0]
        );
    }

    get slotCurrentMask(): string {
        return this.slotSelectedCountry.mask;
    }

    get slotCurrentPlaceholder(): string {
        return this.slotSelectedCountry.placeholder;
    }

    toggleSlotDropdown(): void {
        this.slotIsDropdownOpen = !this.slotIsDropdownOpen;
    }

    selectSlotCountry(option: {
        label: string;
        value: string;
        mask: string;
        placeholder: string;
    }): void {
        this.slotSelectedCountryCode = option.value;
        this.slotIsDropdownOpen = false;
        this.slotCountryPhoneValue = '';
    }

    closeSlotDropdown(): void {
        if (this.slotIsDropdownOpen) {
            this.slotIsDropdownOpen = false;
        }
    }
}

```

Masking Features

- Pattern-Based Input : Define custom input patterns using mask syntax
- Real-time Formatting : Automatic formatting as users type
- Special Character Handling : Control how special characters are processed
- Prefix/Suffix Support : Add currency symbols, units, or other prefixes/suffixes

- Validation Integration : Works seamlessly with existing validation system
- Accessibility : Maintains proper ARIA attributes and keyboard navigation

Common Mask Patterns

- Phone Numbers : (000) 000-0000 for US format
- Dates : 00/00/0000 for MM/DD/YYYY format
- Currency : separator.2 with thousand separators and decimal places
- Credit Cards : 0000 0000 0000 0000 for card number format
- Custom Patterns : Define your own patterns using 0 , 9 , A , S placeholders

States & Validation

Comprehensive validation system with error messages, helper text, and various input states.

```

<aava-textbox
    label="Comments"
    placeholder="Share your thoughts..."
    helper="Please provide detailed feedback to help us improve our service."
    (change)="onTextboxChange($event)"
></aava-textbox>

<aava-textbox
    label="Email"
    placeholder="Enter your email..."
    error="Please enter a valid email address."
    (change)="onTextboxChange($event)"
></aava-textbox>

<aava-textbox
    label="Required Field"
    placeholder="This field is required..."
    [required]="true"
    (change)="onTextboxChange($event)"
></aava-textbox>

---

onTextboxChange(event: Event) {
  const target = event.target as HTMLInputElement;
  console.log('Textbox value changed:', target.value);
}

```

Available States

- Normal - Default input state
- Focused - Active input with enhanced border and shadow
- Disabled - Non-interactive state with dimmed appearance
- Readonly - Display-only state with modified styling
- Error - Validation error state with red styling and error message
- Required - Indicates mandatory fields with asterisk

Validation Features

Error Handling:

- Error messages - Display validation errors with icon
- ARIA compliance - Proper aria-invalid and aria-describedby attributes
- Visual feedback - Error variant styling with red colors
- Icon integration - Alert icons automatically shown with errors

Helper Text:

- Guidance messages - Helpful instructions below input
- Icon support - Info icons automatically shown with helper text
- Conditional display - Helper text hidden when errors are present
- Accessibility - Proper ARIA relationships

Required Fields:

- Visual indicators - Asterisk (*) displayed for required fields
- Label integration - Required indicator integrated with label
- Form validation - Works with Angular form validation

Processing Effects

Advanced processing states with multiple animation options for loading and async operations.

```

<aava-textbox
  [(ngModel)]="processingValue"
  label="Processing State"
  placeholder="Processing..."
  [processing]="true"
  (change)="onProcessingChange($event)"
></aava-textbox>

<aava-textbox
  [(ngModel)]="shimmerValue"
  label="Shimmer Effect"
  placeholder="Validating..."
  [processing]="true"
  processingEffect="shimmer"
  (change)="onShimmerChange($event)"
></aava-textbox>

<aava-textbox
  [(ngModel)]="gradientValue"
  label="Gradient Border"
  placeholder="Submitting..."
  [processing]="true"
  [processingGradientBorder]="true"
  (change)="onGradientChange($event)"
></aava-textbox>

---

processingValue = '';
shimmerValue = '';
gradientValue = '';
customGradientValue = '';
borderPulseValue = '';

onProcessingChange(event: Event) {
  const target = event.target as HTMLInputElement;
  console.log('Processing textbox value changed:', target.value);
}

onShimmerChange(event: Event) {
  const target = event.target as HTMLInputElement;
  console.log('Shimmer textbox value changed:', target.value);
}

onGradientChange(event: Event) {
  const target = event.target as HTMLInputElement;
  console.log('Gradient textbox value changed:', target.value);
}

onCustomGradientChange(event: Event) {
  const target = event.target as HTMLInputElement;
  console.log('Custom gradient textbox value changed:', target.value);
}

onBorderPulseChange(event: Event) {
  const target = event.target as HTMLInputElement;
  console.log('Border pulse textbox value changed:', target.value);
}

```

Available Processing Effects

- border-pulse - Pulsing border animation (default)
- shimmer - Text shimmer effect within input
- gradient-border - Animated multi-color border gradient

Effects System

Modern effects system following Text Input Specifications for consistent visual behavior and accessibility.

Hover Effects

- tint - Subtle color tinting on hover (recommended)
- glow - Enhanced glow effect on hover (for interactive elements)

Processing States

- Default Processing - Border pulse animation with customizable colors
- Shimmer Effect - Text shimmer animation as alternative to border pulse
- Gradient Border - Animated multi-color gradient border for processing state
- Custom Colors - Configurable gradient colors for brand consistency

Accessibility Features

- High Contrast Support - Effects respect high contrast mode settings
- Reduced Motion - Animations respect user's motion preferences
- Focus Indicators - Clear focus states maintained with all effects
- Screen Reader Support - Proper ARIA attributes for all interactive states

API Reference

Inputs

Property	Type	Default	Description
label	string	"	Visible label text above input
placeholder	string	"	Placeholder text shown when empty
variant	TextboxVariant	'default'	Visual variant: 'default' , 'primary' , 'success' , 'error' , 'warning' , 'info'
size	TextboxSize	'md'	Input size: 'xs' , 'sm' , 'md' , 'lg' , 'xl'
disabled	boolean	false	Whether input is disabled

Property	Type	Default	Description
readonly	boolean	false	Whether input is read-only
error	string	"	Error message to display
helper	string	"	Helper text to display
required	boolean	false	Whether input is required
fullWidth	boolean	false	Whether input takes full container width
type	string	'text'	HTML input type
maxlength	number	"	Maximum character length
minlength	number	"	Minimum character length
autocomplete	string	"	HTML autocomplete attribute
id	string	"	Custom element ID
name	string	"	HTML name attribute
icon	string	"	Icon name to display
iconPosition	'start' 'end'	'start'	Position of the icon relative to input
iconSeparator	boolean	false	Whether to show separator between icon and input
iconSpacing	'compact' 'normal' 'relaxed'	'normal'	Icon spacing variant
inputKind	'text' 'phone' 'currency' 'password'	'text'	Type of input for specialized behavior
inputKindLabel	string	"	Label for specialized input types (e.g., country code)
phone	boolean	false	Enable phone input functionality
labelPosition	'start' 'end'	'start'	Position of country prefix label for phone inputs

Masking Properties

Property	Type	Default	Description
mask	string null	null	Input mask pattern (e.g., "(000) 000-0000" for phone)
maskPrefix	string	"	Prefix to add before masked value (e.g., "\$" for currency)
maskSuffix	string	"	Suffix to add after masked value (e.g., "%" for percentage)
maskDropSpecialCharacters	boolean string[]	true	Whether to drop special characters from value
maskShowMaskTyped	boolean	false	Show mask characters as user types
maskThousandSeparator	string	"	Thousand separator character (e.g., "," for numbers)
maskDecimalMarker	'.' ',' ['.','']	'.'	Decimal marker character
maskPatterns	Record	{}	Custom mask patterns for special characters
maskValidation	boolean	false	Enable mask validation
maskAllowNegativeNumbers	boolean	false	Allow negative numbers in numeric masks
maskLeadZeroDateTime	boolean	false	Show leading zeros in date/time masks

Effects System Properties

Property	Type	Default	Description
hoverEffect	'tint' 'glow'	"	Hover effect: Tint (recommended) or Glow

Property	Type	Default	Description
pressedEffect	'solid'	'solid'	Pressed effect: Solid (recommended and only allowed)
processing	boolean	false	Processing state - triggers border pulse by default
processingEffect	'shimmer'	"	Alternative processing effect: Text shimmer animation
processingGradientBorder	boolean	false	Show animated gradient border for processing state
processingGradientColors	string[]	['#e91e63', '#fee140', '#ff9800', '#047857', '#ff9800', '#fee140', '#e91e63']	Colors for processing gradient border
decorativeEffect	'glowBox' 'borderFlow' 'attention' 'wave'	"	Ambient decorative effects for future use
disabledState	'grey'	'grey'	Disabled state appearance: Grey (recommended and only allowed)
customStyles	Record	"	CSS custom properties override for advanced theming

Outputs

Event	Type	Description
iconStartClick	EventEmitter	Emitted when the start icon is clicked
iconEndClick	EventEmitter	Emitted when the end icon is clicked
clickOutSide	EventEmitter	Emitted when a click occurs outside the input
change	EventEmitter	Emitted when the input value changes

Event	Type	Description
blur	EventEmitter	Emitted when the input loses focus
focus	EventEmitter	Emitted when the input gains focus
input	EventEmitter	Emitted on every input event
prefixSelect	EventEmitter<{ label: string; value: string; }>	Emitted when a prefix option is selected
suffixSelect	EventEmitter<{ label: string; value: string; }>	Emitted when a suffix option is selected

Properties

Property	Type	Description
value	string	Current input value
isFocused	boolean	Whether input currently has focus
hasError	boolean	Whether input has error state
hasHelper	boolean	Whether input has helper text

Methods

Method	Parameters	Return Type	Description
setValue()	value: string	void	Set input value programmatically
writeValue()	value: string	void	Set input value (ControlValueAccessor)
registerOnChange()	fn: (value: string) => void	void	Register change callback
registerOnTouched()	fn: () => void	void	Register touched callback
setDisabledState()	isEnabled: boolean	void	Set disabled state

Content Projection Slots

Slot	Description
icon-start	Icons displayed at the start of the input

Slot	Description
icon-end	Icons displayed at the end of the input
prefix	Content displayed before the input text
suffix	Content displayed after the input text

CSS Custom Properties

Property	Description
--textbox-glass-default-background	Background color with glass effect
--textbox-glass-default-blur	Backdrop blur amount for glass effect
--textbox-glass-default-border	Border color for default state
--textbox-glass-default-shadow	Box shadow for glass effect
--textbox-border-radius	Border radius of the input container
--textbox-transition	Transition animation duration
--textbox-input-font	Font properties for input text
--textbox-input-color	Text color for input
--textbox-input-padding	Padding inside the input
--textbox-input-min-height	Minimum height of the input
--textbox-label-font	Font properties for label
--textbox-label-color	Text color for label
--textbox-label-weight	Font weight for label
--textbox-placeholder-color	Color for placeholder text
--textbox-error-color	Color for error messages and state
--textbox-helper-color	Color for helper text
--textbox-icon-color	Color for icons in normal state
--textbox-icon-focus-color	Color for icons when focused
--textbox-variant-default	Colors for default variant
--textbox-variant-primary	Colors for primary variant
--textbox-variant-success	Colors for success variant
--textbox-variant-error	Colors for error variant
--textbox-variant-warning	Colors for warning variant
--textbox-variant-info	Colors for info variant
--textbox-size-xs-padding	Padding for extra small size

Property	Description
--textbox-size-sm-padding	Padding for small size
--textbox-size-md-padding	Padding for medium size
--textbox-size-lg-padding	Padding for large size
--textbox-size-xl-padding	Padding for extra large size
--textbox-size-xs-height	Height for extra small size
--textbox-size-sm-height	Height for small size
--textbox-size-md-height	Height for medium size
--textbox-size-lg-height	Height for large size
--textbox-size-xl-height	Height for extra large size

Best Practices

Design Guidelines

- Choose appropriate variants - Use semantic variants that match the context
- Provide clear labels - Always include descriptive labels for accessibility
- Use helper text wisely - Provide guidance without cluttering the interface
- Handle errors gracefully - Show clear, actionable error messages
- Optimize for touch - Ensure adequate touch targets for mobile users
- Consider glass intensity - Use glass-10 for most cases, glass-50 for emphasis
- Select appropriate sizes - Use xs for very compact layouts, xl for emphasis and accessibility
- Implement masking thoughtfully - Choose mask patterns that match user expectations and data format
- Balance effects and performance - Use effects system for enhanced UX while maintaining performance

Accessibility

- Proper labeling - Label elements correctly associated with inputs
- ARIA attributes - Use aria-invalid , aria-describedby , aria-required
- Error announcement - Error messages announced to screen readers
- Keyboard navigation - Full keyboard support for all interactions
- Focus management - Visible focus indicators and proper focus order
- Icon accessibility - Icons properly labeled and keyboard accessible

Performance

- Validate appropriately - Use client and server validation together
- Debounce input events - For real-time validation and API calls
- Optimize re-renders - Use OnPush change detection strategy
- Efficient icon handling - Load icons efficiently and cache appropriately

Form Integration

- Test with forms - Ensure proper integration with your form handling
- Handle validation - Implement comprehensive validation strategies
- Consider reset behavior - Define clear reset and initial state behavior
- Support reactive forms - Proper ControlValueAccessor implementation

Masking Best Practices

- Choose intuitive patterns - Use mask patterns that users expect (e.g., phone formats)
- Handle edge cases - Consider what happens when users paste or clear masked input
- Provide clear examples - Use placeholders that show the expected format
- Test accessibility - Ensure screen readers can properly announce masked input
- Consider internationalization - Use appropriate separators and formats for different locales
- Balance flexibility - Allow users to edit parts of masked input without losing context